

Proof Nets for Display Logic

Richard Moot

February 2, 2008

1 Introduction

Moot & Puite (2002) have introduced proof nets for the multimodal Lambek calculus $\mathbf{NL}\diamond_{\mathcal{R}}$. Since then, numerous other connectives have been proposed to deal with different linguistic phenomena, a par — or co-tensor, as some authors prefer to call it — together with the corresponding co-implication (Lambek 1993, Moortgat 2007, Bernardi & Moortgat 2007), Galois and dual-Galois connectives (Areces, Bernardi & Moortgat 2001).

We can incorporate these extensions (as well as a few others) into the proof net calculus by simply dropping the restriction that sequents are trees with a unique root node and obtain what are, in effect, proof nets for display logic (Goré 1998). The notion of contraction generalizes to these new connectives without complications.

Like for the Lambek calculus, proof nets for display logic have the advantage of collapsing proofs which differ only for trivial reasons. The display rules in particular are compiled away in the proof net representation.

2 Proof Nets

Proof nets are an optimal representation for proofs of linear logic introduced by Girard (1987).

2.1 Links and Proof Structures

Definition 1 A link, as defined by Moot & Puite (2002)¹, is a tuple $\langle \tau, P, Q, m \rangle$ where

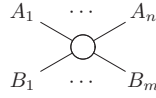
- τ , the type of the link, is either \otimes or \wp ,
- P is a list of premisses A_1, \dots, A_n ,

¹Some of the details are slightly different: the rule name ν has been suppressed since we need only the mode part of it and the subsequences p and q have been replaced by the main formula argument m

- Q is a list of conclusions B_1, \dots, B_m ,
- m , the main formula of the link, is either ϵ or a member of $P \cup Q$.

If $m = \epsilon$ then we will call the link *neutral*, if it is a member of P we will call the link a *left link* and if it is a member of Q we will call it a *right link*.

We draw links as shown below, with the premisses from left to right above the link and the conclusions below it.



Visually, we distinguish between tensor links — which we draw with a white circle at the interior — and par links — which are drawn with a black circle. Finally, unless $m = \epsilon$ we denote the main formula of the link by drawing an arrow from the center of the link to this formula. In this case, we will refer to the other formulas as the *active formulas* of the link.

This definition of link allows us to create quite a number of links in addition to the ones given in that article. The links there were all possible unary and binary links given the assumption of a unique conclusion for every tensor link. Once we drop this constraint, different types of link become possible.

Figure 1 gives an overview of the 9 different forms of tensor links of arity 2 or less (2 nullary, 3 unary and 4 binary), together with the logical connectives associated with their different ports for a total of 2 nullary, 6 unary and 12 binary connectives.

Note that — as displayed in the figure — none of the tensor links have a main formula according to Definition 1. However, in case we need to find the main and active formulas of a link, we can do so by simply inspecting the formulas assigned to the different ports.

Corresponding to each tensor link is a par link which is a ‘mirror image’ of the corresponding tensor link as shown in Figure 2.

If we want to make more distinctions, we can use modes — as is usual in multimodel categorical grammar (Moortgat 1997) — like we did in (Moot & Puite 2002) for $\mathbf{NL} \diamond_{\mathcal{R}}$ and write the mode in the circle of the link. To somewhat reduce the (already extensive) vocabulary, we will not talk about modes in this article, but the current approach can be extended to incorporate them without problems. Adding them would just amount to inserting mode information in all tensor and par links and demanding identity between the two modes to allow a contraction.

Definition 2 A proof structure $\langle S, \mathcal{L} \rangle$ is a finite set of formulas S together with a set of links \mathcal{L} as shown in Figures 1 and 2 such that.

- every formula of S is at most once the premiss of a link.

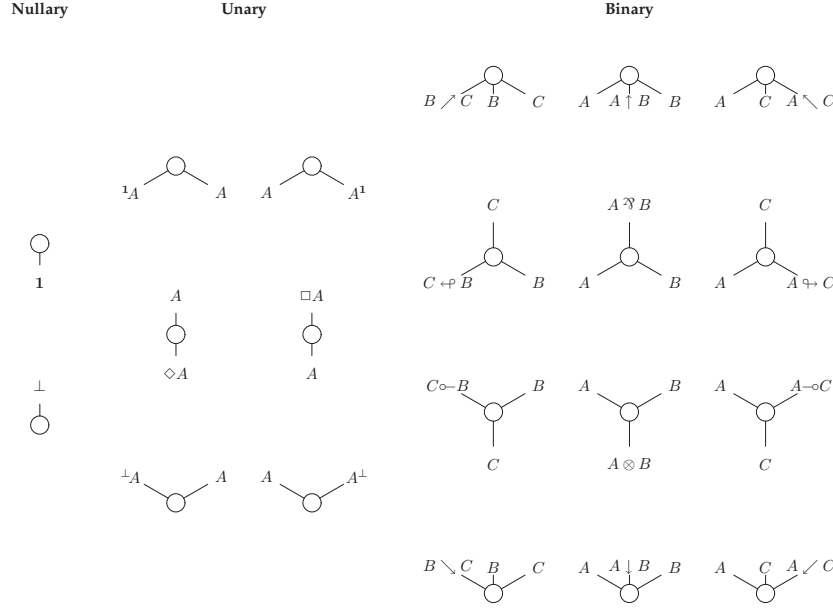


Figure 1: All tensor links of arity 2 or less

- every formula of S is at most once the conclusion of a link.

Formulas which are not the conclusion of any link are the hypotheses H of the proof structures, whereas the formulas which are not the premiss of any link are the conclusions C .

Readers familiar with proof nets from linear logic will note the absence or cut and axiom links. We have axiom and cut *formulas* instead.

Definition 3 An axiom formula is a formula which is not the main formula of any link. A cut formula is a formula which is the main formula of two links.

Figure 3 shows the proof structure for $(\perp \multimap A) \multimap \perp \vdash A$ on the left. The A formula is the only axiom in the structure.

There are some differences in the notation of other authors. It is closest to display logic, with \wp taking the place of \oplus and \perp taking the place of 0 , much in the spirit of the connectives from linear logic. The symbols for the two implications \multimap and \multimap have been chosen to remind us they are the residuals of \wp . Table 7 in Appendix A gives an overview of the logical symbols used and the corresponding logical symbols in various other logics.

2.2 Abstract Proof Structures

From a proof structure we obtain an *abstract* proof structure simply by erasing all formulas on the internal nodes. We only keep the formulas on the premisses

- every vertex of V is at most once the premiss of a link,
- every vertex of V is at most once the conclusion of a link

p is a labelling function assigning a formula to the hypotheses of the abstract proof structure, that is, to those formulas which are not the conclusion of any link,

q is a labelling function assigning a formula to the conclusions of the abstract proof structure, that is, to those formulas which are not the premiss of any link.

We will draw the nodes of abstract proof structures as shown below

$$\begin{array}{c} H \\ \vdots \\ C \end{array}$$

where H is the hypothesis assigned to this node and C is the conclusion assigned to it. Both H and C can be empty.

Figure 3 shows the abstract proof structure corresponding to the proof structure of $(\perp \multimap A) \multimap \perp \vdash A$ on the right.

Definition 5 A tensor tree is an acyclic connected abstract proof structure containing only tensor links.

We say a tensor tree with hypotheses A_1, \dots, A_n and conclusions B_1, \dots, B_m corresponds to $A_1, \dots, A_n \vdash B_1, \dots, B_m$. However, in order to determine the structure of the sequent to which a tensor tree corresponds, we first have to do a bit of work.

2.3 Sequents and Tensor Trees

An advantage of the formulation of Moot & Puite (2002) was that, because of the shape of the two tensor links we considered and because of the conditions on proof structures, a tensor tree was a rooted tree. The new types of tensor links do not preserve this property. Figure 4 shows an example.

Here, we have three premisses (A , B and C) and two conclusions (D and E) but they are grouped in such a way that we cannot turn them into a sequent $A, B, C \vdash D, E$ straightforwardly.

To solve this problem, we abolish the notion that the premisses of a sequent are on the left hand side of the turnstile and the conclusions on the right hand side. We simply split the tensor tree at an arbitrary point and translate the two trees we obtain into sequents in such a way that we can uniquely recover the original tensor tree.

Figure 5 lists the structural connectives we need: 1 nullary, 3 unary and 6 binary. The structural connectives are essentially borrowed from display logic.

Definition 6 Let T be a tensor tree and x be a node on this tensor tree, the sequent $T(x)$ is defined as follows. We split T at x to obtain a tree T_h^x which has x as a hypothesis and a tree T_c^x which has x as a conclusion. Without changing the shape of

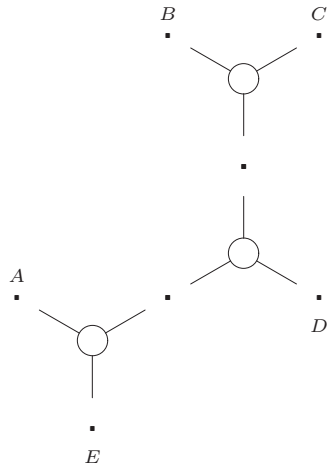


Figure 4: A tensor tree which is not rooted

Nullary

Unary

Binary

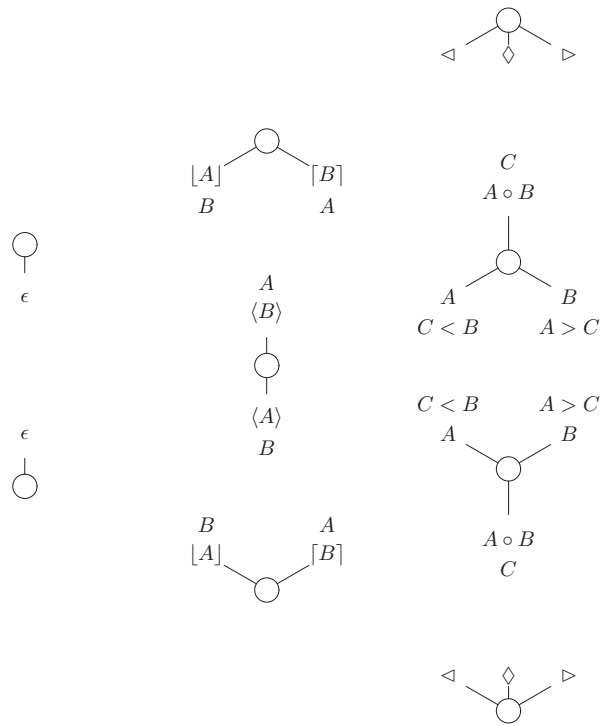


Figure 5: Information Flow

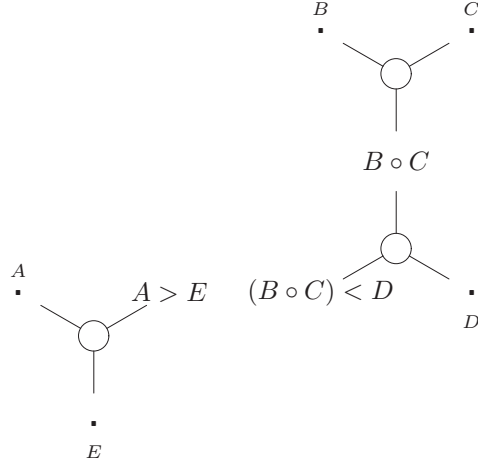


Figure 6: Computing the flow

$$\begin{array}{c}
\frac{\frac{[\Gamma] \vdash \Delta}{[\Delta] \vdash \Gamma} [dgc] \quad \frac{\Gamma \vdash [\Delta]}{\Delta \vdash [\Gamma]} [gc]}{\frac{\langle \Gamma \rangle \vdash \Delta}{\Gamma \vdash \langle \Delta \rangle} [rc]} \\
\frac{\frac{\Gamma_1 \vdash \Delta < \Gamma_2}{\Gamma_1 \circ \Gamma_2 \vdash \Delta} [rc] \quad \frac{\Gamma < \Delta_2 \vdash \Delta_1}{\Gamma \vdash \Delta_1 \circ \Delta_2} [drc]}{\frac{\Gamma_2 \vdash \Gamma_1 > \Delta}{\Delta_1 > \Gamma \vdash \Delta_2} [drc]} \\
\frac{\frac{\Delta \triangleright \Gamma_2 \vdash \Gamma_1}{\Gamma_1 \diamond \Gamma_2 \vdash \Delta} [dgc] \quad \frac{\Delta_1 \vdash \Gamma \triangleleft \Delta_2}{\Gamma \vdash \Delta_1 \diamond \Delta_2} [gc]}{\frac{\Gamma_1 \triangleright \Delta \vdash \Gamma_2}{\Delta_2 \vdash \Delta_1 \triangleright \Gamma} [gc]}
\end{array}$$

Table 1: Sequent Rules — Display Rules

either of the trees, we will consider the two instances x as the root of their respective trees and all other hypotheses and conclusions as its leaves. Moving from these leaves towards x we use the flow of Figure 5 to compute a term S_c for T_c^x and a term S_h for T_h^x . The final sequent $T(x)$ is $S_c \vdash S_h$.

Note that the tree upwards of the split point becomes the antecedent, while the tree down from it becomes the succedent. Figure 6 shows an example of computing the flow corresponding a split vertex.

We see that, depending on our choice of the ‘split point’ of the tensor tree, Figure 4 corresponds to one of the following sequents, of which we computed the second in Figure 6.

$$\begin{aligned}
& A \vdash E < ((B \circ C) < D) \\
& (B \circ C) < D \vdash A > E \\
& A \circ ((B \circ C) < D) \vdash E \\
& B \circ C \vdash (A > E) \circ D \\
& B \vdash ((A > E) \circ D) < C \\
& C \vdash B > ((A > E) \circ D) \\
& (A > E) > (B \circ C) \vdash D
\end{aligned}$$

There is exactly one possible sequent for each vertex in the graph; this is no coincidence as it corresponds to a ‘display property’ for each vertex. Note that all of these sequents are interderivable thanks to the display rules of Table 1. In the following, to make it easier to refer to each of the display rules, we will write the two structural connectives between parentheses. For example, we will write $rc(<; >)$ for a replacement (from top to bottom) for a $<$ structural connective by a $>$ structural connective.

Lemma 7 *Let \mathcal{T} be a tensor tree and x and y two nodes on this tree. Take $s = \mathcal{T}(x)$ and $s' = \mathcal{T}(y)$. s and s' are equivalent up to the display rules.*

Proof Induction on the length l of the unique path between x and y . In case $l = 0$ then s and s' are identical.

Suppose $l > 0$, induction hypothesis ... we essentially replace one structural connective by another corresponding to either $[rc]$, $[drc]$, $[gc]$ or $[rc]$... \square

Lemma 8 *If \mathcal{T} is a tensor tree containing one or more links, then at least one of the leaves of the corresponding proof structure (hypotheses and conclusions) is the main formula of its link.*

Proof Assume \mathcal{T} has n leaves and that $n - 1$ of these leaves are the active formulas of their link. We show that the last leaf l must be the main formula of its link. Without changing the orientation of any of the links, we can see \mathcal{T} as a tree with root l and with $n - 1$ leaves. Working our way upward from the deepest level d towards the root we show that the nodes at the next level are always the active formulas of their link. This means that when we arrive at the last link of which the root is one of its ports, all its other ports are the active formulas of the link, which means the root node must be the main formula. \square

2.4 Sequent Rules

In addition to the display rules, which allow us to turn any formula in the sequent to either the complete left hand side or the complete right hand side of a sequent, we have a left and right rule for each of the connectives.

Given the display property we can always assume — as shown by the rules in Tables 2 to 5 — that the context is on the other side of the sequent as the

$$\begin{array}{c}
\frac{\epsilon \vdash \Delta}{1 \vdash \Delta} [L1] \quad \frac{}{\epsilon \vdash 1} [R1] \\
\frac{}{\perp \vdash \epsilon} [L \perp] \quad \frac{\Gamma \vdash \epsilon}{\Gamma \vdash \perp} [R \perp]
\end{array}$$

Table 2: Sequent Rules — Nullary Connectives

$$\begin{array}{c}
\frac{\Delta \vdash A}{A^\perp \vdash [\Delta]} [L.\perp] \quad \frac{\Gamma \vdash [A]}{\Gamma \vdash A^\perp} [R.\perp] \\
\frac{\Delta \vdash A}{\perp A \vdash [\Delta]} [L^\perp.] \quad \frac{\Gamma \vdash [A]}{\Gamma \vdash \perp A} [R^\perp.] \\
\frac{[A] \vdash \Delta}{A^1 \vdash \Delta} [L.^1] \quad \frac{A \vdash \Gamma}{[\Gamma] \vdash A^1} [R.^1] \\
\frac{[A] \vdash \Delta}{^1 A \vdash \Delta} [L^1.] \quad \frac{A \vdash \Gamma}{[\Gamma] \vdash ^1 A} [R^1.] \\
\frac{\langle A \rangle \vdash \Delta}{\diamond A \vdash \Delta} [L\diamond] \quad \frac{\Gamma \vdash A}{\langle \Gamma \rangle \vdash \diamond A} [R\diamond] \\
\frac{A \vdash \Delta}{\Box A \vdash \langle \Delta \rangle} [L\Box] \quad \frac{\Gamma \vdash \langle A \rangle}{\Gamma \vdash \Box A} [R\Box]
\end{array}$$

Table 3: Sequent Rules — Unary Connectives

logical connective we would want to treat. Apart from the binary galois and dual galois connectives, which I haven't seen elsewhere, these rules are the same up to notational choices as those of display logic.

2.5 Contractions

A tensor and a par link contract when the tensor link is connected — respecting up/down and left/right — to the par link at all its ports except the single main port of the par link and the corresponding port of the tensor link.

The redex for all contraction is a single node as follows.

$$\begin{array}{c}
H \\
\bullet \\
C
\end{array}$$

Both links and the internal nodes will be removed from the resulting graph and the two exterior nodes will be merged, inheriting the hypothesis and conclusion label of the nodes in case either node is a hypothesis or conclusion of the abstract proof structure.

Figure 7 shows the contractions for the binary residuated connectives \multimap , \otimes and $\circ\multimap$. These are exactly the contractions proposed for NL.

$\frac{A \circ B \vdash \Delta}{A \otimes B \vdash \Delta} [L\otimes]$	$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1 \circ \Gamma_2 \vdash A \otimes B} [R\otimes]$
$\frac{\Gamma \vdash A \quad B \vdash \Delta}{A \multimap B \vdash \Gamma > \Delta} [L\multimap]$	$\frac{\Gamma \vdash A > B}{\Gamma \vdash A \multimap B} [R\multimap]$
$\frac{A \vdash \Delta \quad \Gamma \vdash B}{A \multimap B \vdash \Delta < \Gamma} [L\multimap-]$	$\frac{\Gamma \vdash A < B}{\Gamma \vdash A \multimap B} [R\multimap-]$
$\frac{A \vdash \Delta_1 \quad B \vdash \Delta_2}{A \wp B \vdash \Delta_1 \circ \Delta_2} [L\wp]$	$\frac{\Gamma \vdash A \circ B}{\Gamma \vdash A \wp B} [R\wp]$
$\frac{A > B \vdash \Delta}{A \wp B \vdash \Delta} [L\wp\rightarrow]$	$\frac{A \vdash \Delta \quad \Gamma \vdash B}{\Delta > \Gamma \vdash A \wp B} [R\wp\rightarrow]$
$\frac{A < B \vdash \Delta}{A \wp B \vdash \Delta} [L\wp\leftarrow]$	$\frac{\Gamma \vdash A \quad B \vdash \Delta}{\Gamma < \Delta \vdash A \wp B} [R\wp\leftarrow]$

Table 4: Sequent Rules — Binary Connectives

$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{A \downarrow B \vdash \Gamma_1 \diamond \Gamma_2} [L\downarrow]$	$\frac{\Gamma \vdash A \diamond B}{\Gamma \vdash A \downarrow B} [R\downarrow]$
$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{A \swarrow B \vdash \Gamma_1 \triangleleft \Gamma_2} [L\swarrow]$	$\frac{\Gamma \vdash A \triangleleft B}{\Gamma \vdash A \swarrow B} [R\swarrow]$
$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{A \searrow B \vdash \Gamma_1 \triangleright \Gamma_2} [L\searrow]$	$\frac{\Gamma \vdash A \triangleright B}{\Gamma \vdash A \searrow B} [R\searrow]$
$\frac{A \diamond B \vdash \Delta}{A \uparrow B \vdash \Delta} [L\uparrow]$	$\frac{A \vdash \Delta_1 \quad B \vdash \Delta_2}{\Delta_1 \diamond \Delta_2 \vdash A \uparrow B} [R\uparrow]$
$\frac{A \triangleleft B \vdash \Delta}{A \searrow B \vdash \Delta} [L\searrow]$	$\frac{A \vdash \Delta_1 \quad B \vdash \Delta_2}{\Delta_1 \triangleleft \Delta_2 \vdash A \searrow B} [R\searrow]$
$\frac{A \triangleright B \vdash \Delta}{A \swarrow B \vdash \Delta} [L\swarrow]$	$\frac{A \vdash \Delta_1 \quad B \vdash \Delta_2}{\Delta_1 \triangleright \Delta_2 \vdash A \swarrow B} [R\swarrow]$

Table 5: Sequent Rules — Binary Connectives (continued)

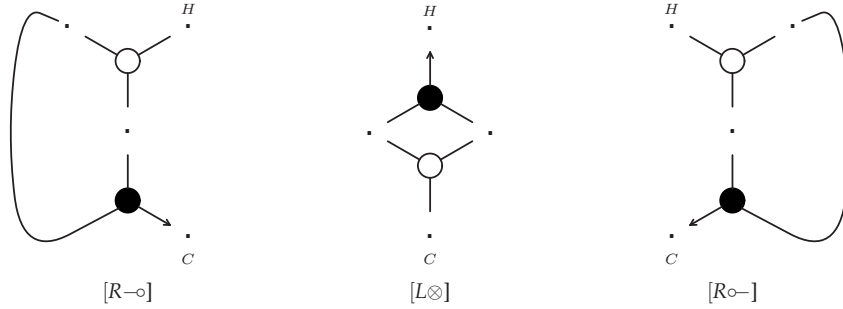


Figure 7: Contractions: binary residuated

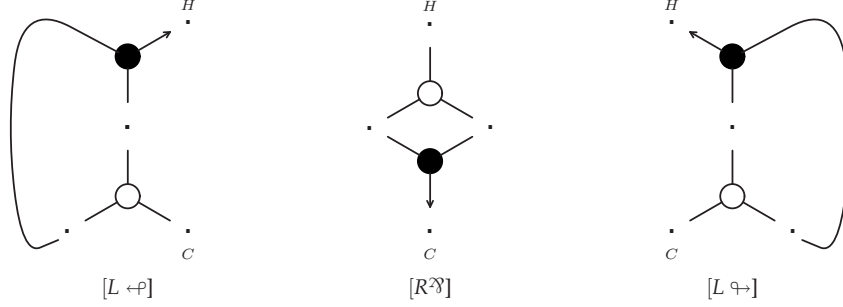


Figure 8: Contractions: binary dual residuated

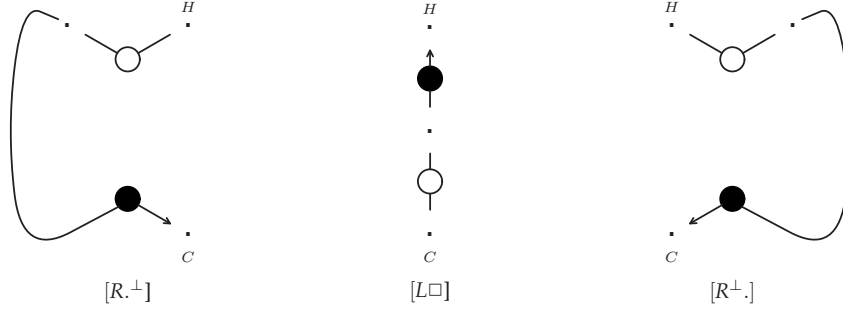


Figure 9: Contractions: unary 1

Figure 8 shows their duals: the contractions for \nwarrow , \bowtie and $\leftarrow\oplus$. They are obtained from the residuated conversions by mirroring the figures on the x-axis and by exchanging hypotheses and conclusions.

Figures 9 and 10 show the unary contractions for the Galois, residuated and dual Galois connectives. They are obtained from Figures 7 and 8 by removing one up-down connection from each of the redexes.

Figure 11 shows the nullary contractions. Since the nullary links have just one port, the condition that all ports except the main port have to be connected is satisfied trivially, thus the nullary contractions simply identify different nodes in the graph.

2.6 Structural Rules

We can extend the proof net as well as the sequent calculus with an arbitrary number of structural conversions. A structural conversion in the proof net calculus is simply a rewrite of one tensor tree into another in such a way that both trees have the same hypotheses and the same conclusions, though we are

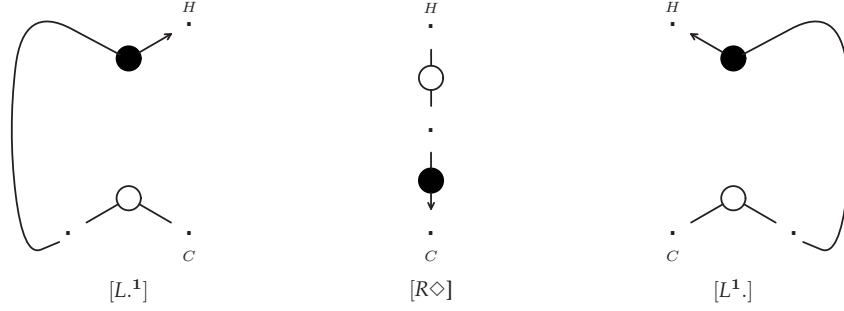


Figure 10: Contractions: unary 2

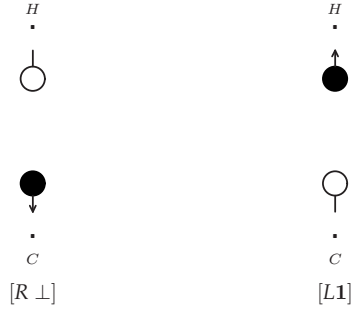


Figure 11: Contractions: nullary

allowed to change their order. Figure 12 shows the schematic form of a structural conversion. The x vertices are the n hypotheses of the conversion, the y vertices the m conclusions and π (resp. π') is a permutation of the hypotheses (resp. the conclusions).

This restriction means the contraction and weakening rules are not allowed:

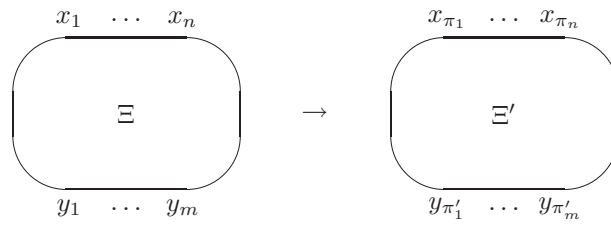


Figure 12: Schematic Form of a Structural Conversion

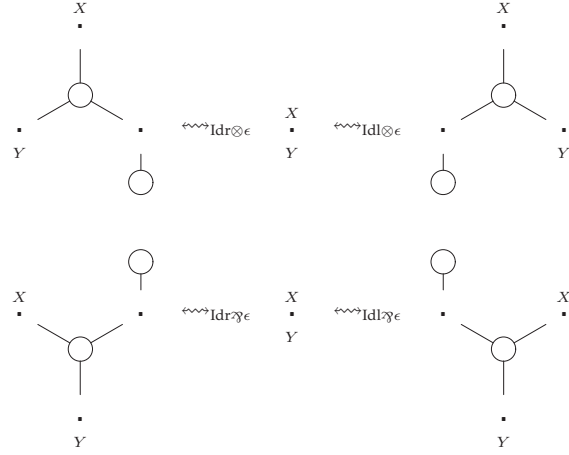


Figure 13: Identity rules for tensor and par

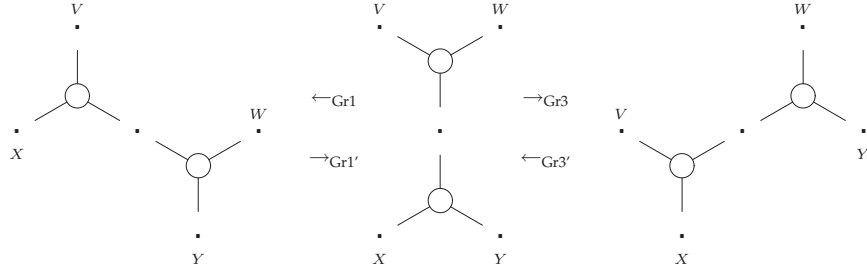


Figure 14: Grishin Rules: Mixed Associativity

we operate essentially in a fragment of multiplicative linear logic (Girard 1987).

Figures 13 and 14 shows some well-known examples of valid structural rules which don't change the order of the hypotheses and premisses.

The structural rules of Figure 14 play a role similar to the mixed associativity rules in multimodal system. The corresponding mixed commutativity rules are shown in Figure 15. We will refer to the primed versions of the structural conversions, ie. those moving *towards* the center structure, as the Grishin class I rules, whereas the non-primed rules, those moving *away from* the center structure are the Grishin class IV rules (Grishin 1983).

Given a structural conversion, what is the sequent rule which corresponds to it? As shown by the following lemma, there are multiple equivalent possibilities, depending on which of the leaves is displayed.

Definition 9 Let s be a structural conversion and l one of its leaves. $s(l)$ will denote

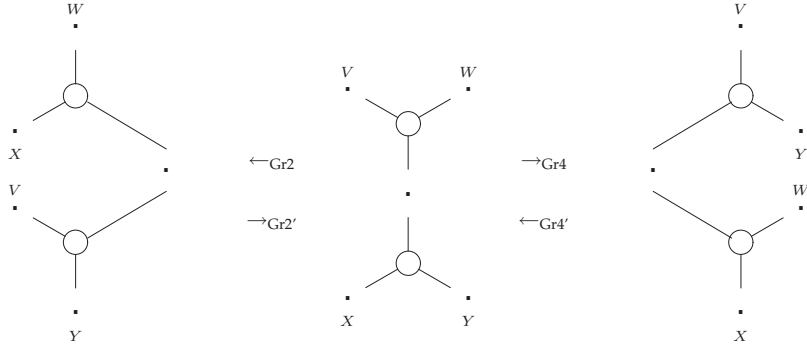


Figure 15: Grishin Rules: Mixed Commutativity

the structural rule obtained by computing the flow according to Definition 6 with the exception that every hypothesis leaf x_i will correspond to a structural variable Γ_i and every conclusion leaf y_i to a structural variable Δ_i .

For example, depending on whether we use hypothesis X or conclusion Y to obtain a corresponding structural rule, we obtain either rule $[Idr \otimes \epsilon 1]$ or rule $[Idr \otimes \epsilon 2]$.

$$\frac{\Gamma \circ \epsilon \vdash \Delta}{\Gamma \vdash \Delta} [Idr \otimes \epsilon 1] \quad \frac{\Gamma \vdash \Delta < \epsilon}{\Gamma \vdash \Delta} [Idr \otimes \epsilon 2]$$

Note that the two rules are equivalent.

$$\frac{\frac{\Gamma \vdash \Delta < \epsilon}{\Gamma \circ \epsilon \vdash \Delta} [rc \circ <]}{\Gamma \vdash \Delta} [Idr \otimes \epsilon 1] \quad \frac{\frac{\Gamma \circ \epsilon \vdash \Delta}{\Gamma \vdash \Delta < \epsilon} [rc < \circ]}{\Gamma \vdash \Delta} [Idr \otimes \epsilon 2]$$

Lemma 10 Let l_1 and l_2 be two distinct leaves of a structural conversion s . Then the structural rules $s(l_1)$ and $s(l_2)$ are interderivable using only the other rule and the display rules.

Proof Similar to the proof of Lemma 7 we follow the unique path from l_1 to l_2 applying a display rule at each step to derive $s(l_2)$ from $s(l_1)$. Given that all the display rules are reversible we can derive $s(l_1)$ from $s(l_2)$ using the inverse rules. \square

Example 11 Using $Gr1'$, $Gr1$ and the right identity for tensor and left identity for par, we can derive $(\perp \multimap A) \multimap \perp \vdash A$. Figures 16 to 19 show how the abstract proof structure of Figure 3 can be contracted.

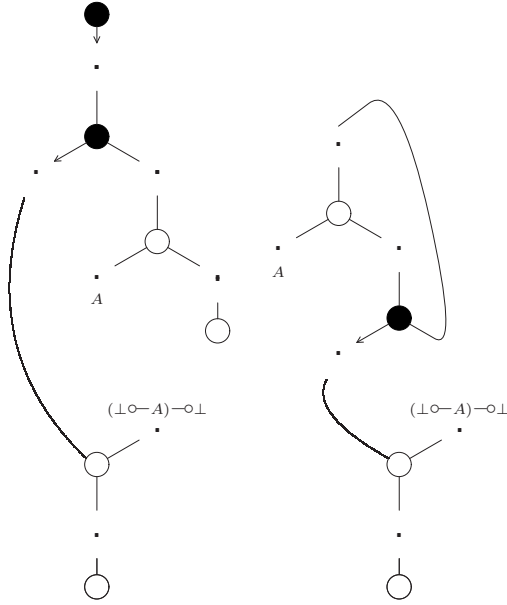


Figure 16: Right identity for \otimes , followed by the \perp contraction

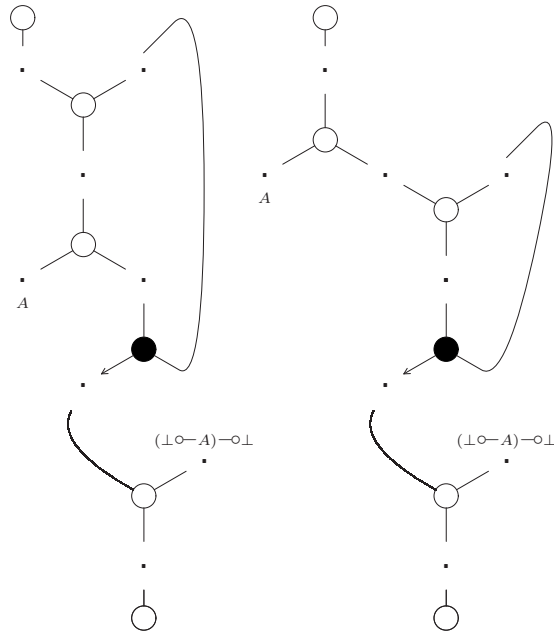


Figure 17: Left identity for par followed by Gr1

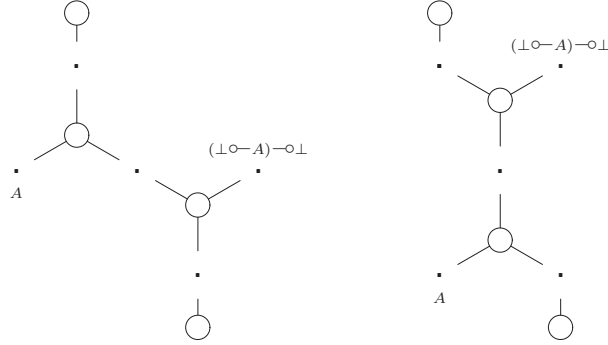


Figure 18: The \multimap contraction followed by Gr1'

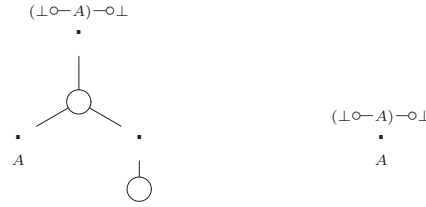


Figure 19: Left identity for par and right identity for tensor

3 Correctness

We are now in a position to prove the main theorem: that derivability in the sequent calculus and contractability in the proof net calculus coincide.

Theorem 12 *A proof structure \mathcal{S} is correct (ie. corresponds to a sequent proof of $\Gamma \vdash \Delta$) if and only if its abstract proof structure \mathcal{A} converts to a tensor tree of $\Gamma \vdash \Delta$.*

\Rightarrow Suppose π is a sequent calculus proof of $\Gamma \vdash \Delta$. We construct a proof structure together with a reduction sequence ρ reducing it to tensor tree $\Gamma \vdash \Delta$ by induction on the depth d of π .

If $d = 1$ then π is one of the axioms. We consider each case separately.

If the conclusion of the sequent is $\perp \vdash \epsilon$ then the corresponding proof structure and abstract proof structure look as shown below.



Note how this is a proof net of $\perp \vdash \epsilon$ as required.

Similarly, if the conclusion of the axiom is $\epsilon \vdash 1$ then the corresponding proof structure and abstract proof structure form a proof net of this sequent as shown below.

$$\begin{array}{ccc} \text{○} & \rightarrow & \text{○} \\ | & & | \\ \mathbf{1} & & \cdot \\ & & \mathbf{1} \end{array}$$

Finally, if the conclusion of the axiom is $A \vdash A$ for some formula A then we are in the following situation

$$\begin{array}{ccc} A & \rightarrow & \begin{array}{c} A \\ \cdot \\ A \end{array} \end{array}$$

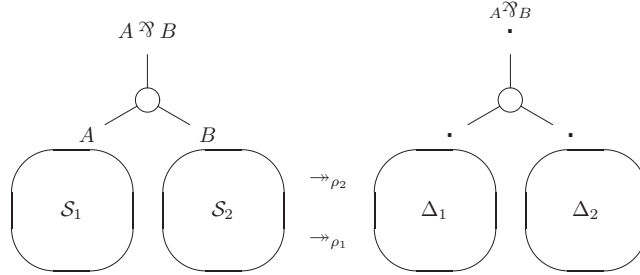
which is a proof net of $A \vdash A$.

If $d > 1$ then we look at the last rule in the proof. Suppose it is a $[L^{\wp}]$ rule.

$$\frac{\begin{array}{c} \vdots \pi_1 \\ A \vdash \Delta_1 \end{array} \quad \begin{array}{c} \vdots \pi_2 \\ B \vdash \Delta_2 \end{array}}{A \wp B \vdash \Delta_1 \circ \Delta_2} [L^{\wp}]$$

Given that both π_1 and π_2 have a depth smaller than d , we can apply to induction hypothesis to obtain a proof structure S_1 with hypothesis A which reduces to a tensor tree of $A \vdash \Delta_1$ by reduction sequence ρ_1 and a proof structure S_2 with hypothesis B which reduces to a tensor tree of $B \vdash \Delta_2$ by reduction sequence ρ_2 .

We can combine these two proof nets as shown below.

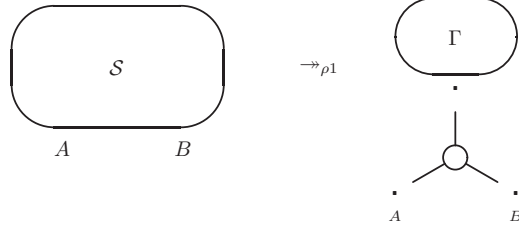


Note that, since ρ_1 and ρ_2 operate on different parts of the resulting abstract proof structure, any interleaving of ρ_1 and ρ_2 will provide a valid reduction sequence ρ producing a proof net of $A \wp B \vdash \Delta_1 \circ \Delta_2$.

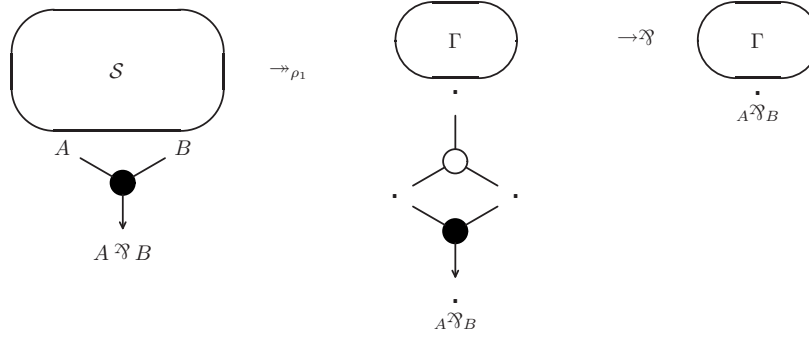
Suppose the last rule is a $[R^{\wp}]$ rule.

$$\frac{\begin{array}{c} \vdots \pi_1 \\ \Gamma \vdash A \circ B \end{array}}{\Gamma \vdash A \wp B} [R^{\wp}]$$

Given that π_1 has a depth of $d - 1$ we can apply the induction hypothesis to give us a proof structure with conclusions A and B which converts to a tensor tree of $\Gamma \vdash A \wp B$ by a conversion sequence ρ_1 .



We can add the par link for $A \wp B$ to the proof structure above after which the reduction sequent ρ_1 produces a redex for the $[R\wp]$ contraction. We append this contraction at the end of ρ_1 to produce the final contraction sequence ρ , producing a proof net of $\Gamma \vdash A \wp B$ as required.



The other logical rules are similar and easily verified.

\Leftarrow Suppose \mathcal{A} corresponding to S converts to a tensor tree \mathcal{T} by means of a conversion sequence s . We proceed by induction on the length l of s to construct a sequent proof of \mathcal{T} . In what follows, I will often use ‘a derivation d of $\Gamma \vdash \Delta$ ’ where it would be more precise but also more cumbersome to use ‘a derivation d which can be extended using only the display rules to a derivation of $\Gamma \vdash \Delta$. I trust this will not lead to confusion.

If $l = 0$ then $\mathcal{A} = \mathcal{T}$. We proceed by induction on the number of connectors c in \mathcal{T} .

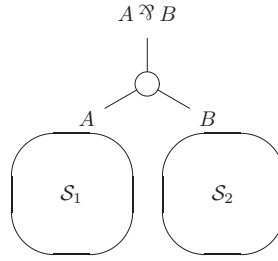
If $c = 0$ then S and \mathcal{T} are of the form

$$A \rightarrow \begin{array}{c} A \\ \bullet \\ A \end{array}$$

which corresponds to the sequent proof

$$\frac{}{A \vdash A} [Ax]$$

If $c > 0$ then by Lemma 8, we know that S has a formula which is the main leaf of its link, call it D . We proceed by case analysis. If D is of the form $A \wp B$, then we are in the following situation.



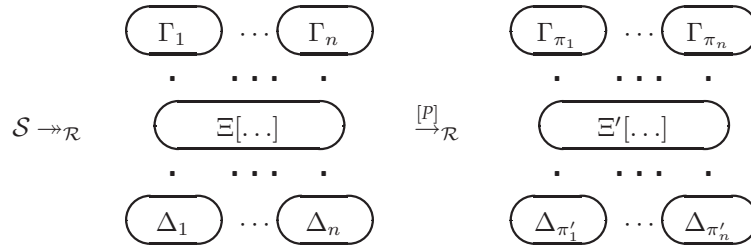
Because the proof structure is a tree, the par link separates the structure into two parts: S_1 with hypothesis A and S_2 with hypothesis B . By induction hypothesis, there are derivations d_1 of $A \vdash \Delta_1$ and d_2 of $B \vdash \Delta_2$, which we can combine as follows.

$$\frac{A \vdash \Delta_1 \quad B \vdash \Delta_2}{A \wp B \vdash \Delta_1 \circ \Delta_2} [L\wp]$$

The other cases are similar.

Suppose now $l > 0$. We look at the last conversion.

Suppose the last conversion is a structural conversion, then we are schematically in the following situation.



Given that we have a structural conversion $[P]$ we know there is at least one structural rule which corresponds to it, where one of the leaves of both Ξ and Ξ' is displayed. In case this leaf is a hypothesis of Ξ (assume it's Γ_i), induction hypothesis gives us a proof d which we can extend as follows.

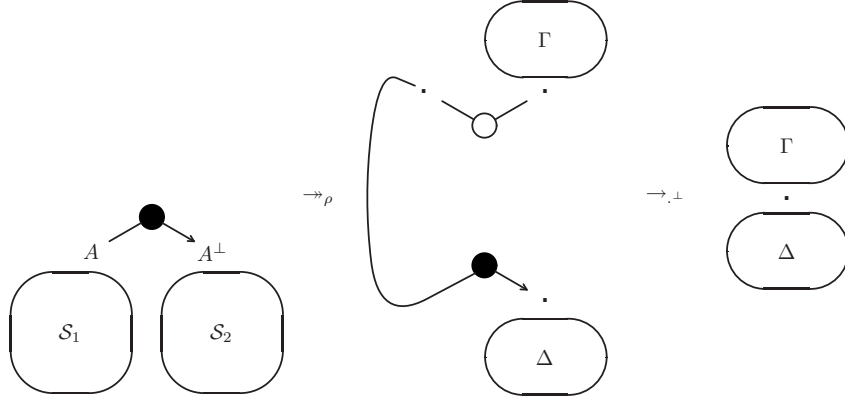
$$\frac{\begin{array}{c} \vdots d \\ \Gamma_i \vdash \Xi \end{array}}{\Gamma_i \vdash \Xi'} [P]$$

In case the leaf is a conclusion Δ_i we operate symmetrically, obtaining.

$$\frac{\begin{array}{c} \vdots d \\ \Xi \vdash \Delta_i \end{array}}{\Xi' \vdash \Delta_i} [P]$$

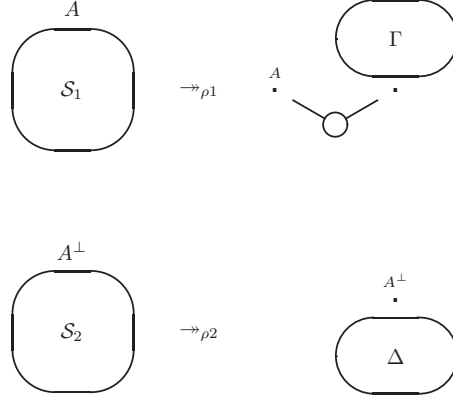
Suppose the last conversion is a contraction. We proceed by case analysis.

$[R.^\perp]$ In case the last conversion is a $.^\perp$ contraction we are schematically in the following situation.



Looking backwards from the endsequent, the par link forms a barrier: every structural rewrite has to be performed either fully in Γ — where it will finally end up producing S_1 — or fully in Δ — where it will finally end up producing S_2 . From this perspective, every contraction simply expands a single node and is therefore performed in just one of the two substructures as well. Therefore, we can separate the conversions of ρ into those which are fully in S_1 reducing it to $\Gamma \vdash [A]$ and those which are fully in S_2 reducing it to $A^\perp \vdash \Delta$. We will call these two reduction sequences ρ_1 and ρ_2 respectively.

Removing the par link from the figure above gives us the following two proof structures with their corresponding reduction sequences.

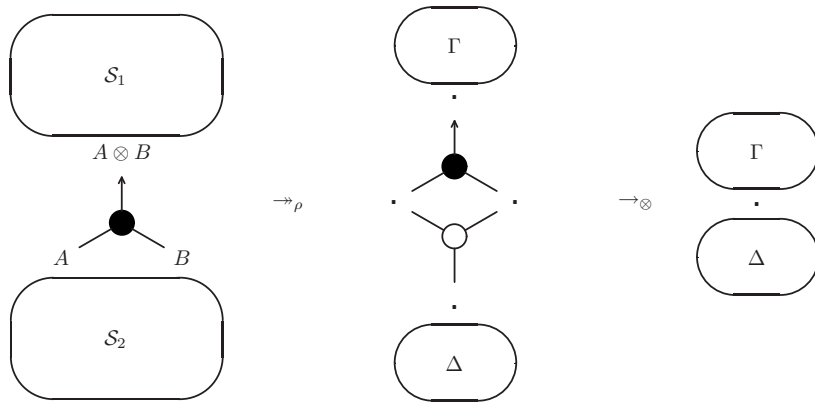


Since the length of $\rho_1 + \rho_2$ is less than the length of ρ — the final contraction being removed — we can apply the induction hypothesis to give us a proof d_1 of $\Gamma \vdash [A]$ and a proof d_2 of $A^\perp \vdash \Delta$. We can combine these two proofs into a proof of $\Gamma \vdash \Delta$ as follows.

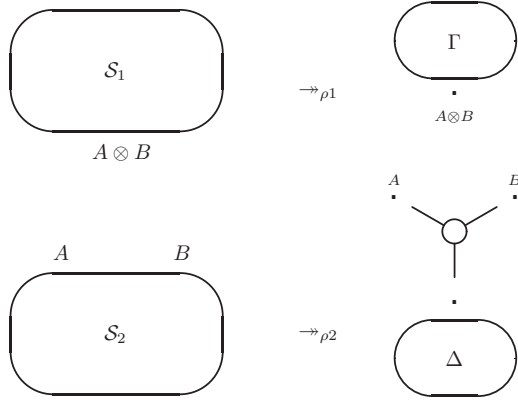
$$\frac{\frac{\vdots d_1}{\Gamma \vdash [A]} [R.\perp] \quad \frac{\vdots d_2}{A^\perp \vdash \Delta} [Cut]}{\Gamma \vdash \Delta} [R.\perp]$$

$[R.\perp]$ Symmetric.

$[L\otimes]$ In case the last contraction is a \otimes contraction, the proof structure and the conversion sequence for the corresponding abstract proof structure look as shown below.



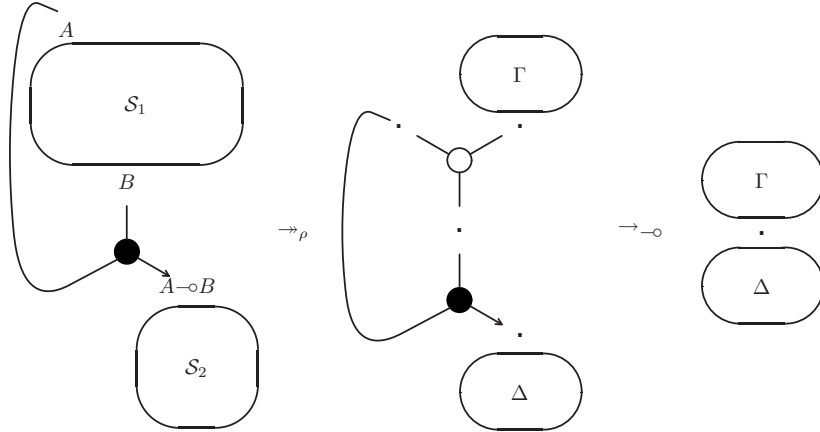
We again eliminate the par link and its contraction and partition the remaining conversions over two disjoint sequences as shown below.



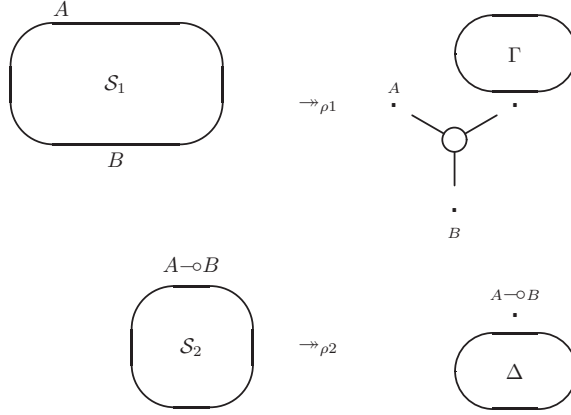
Now the induction hypothesis gives us a derivation d_1 of $\Gamma \vdash A \otimes B$ and a derivation d_2 of $A \circ B \vdash \Delta$. We combine these two derivations into a derivation of $\Gamma \vdash \Delta$ as follows.

$$\frac{\frac{\Gamma \vdash A \otimes B \quad \frac{A \circ B \vdash \Delta}{A \otimes B \vdash \Delta} [L\otimes]}{\Gamma \vdash \Delta} [Cut]$$

$[R-\multimap]$ If the last contraction is a \multimap contraction, the proof structure and reduction sequence look as follows.



As before we remove the par link and its contraction and separate the conversion sequences which are in Γ and Δ . The result is shown below.

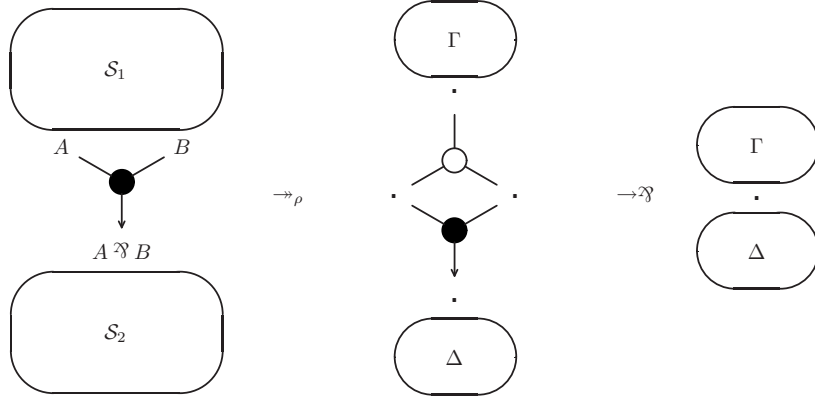


Induction hypothesis now gives us a derivation d_1 from S_1 to $\Gamma \vdash A > B$ and a derivation d_2 from S_2 to $A \multimap B \vdash \Delta$. We can combine these proofs in the following way.

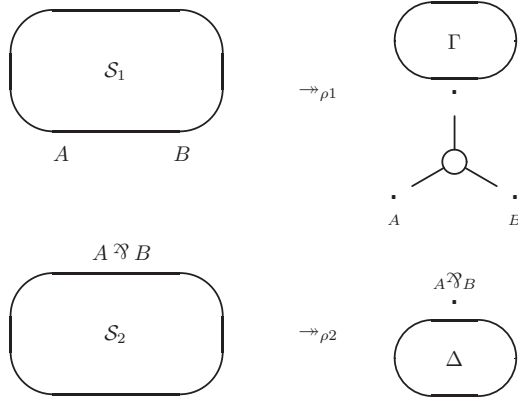
$$\frac{\frac{\frac{\vdots d_1}{\Gamma \vdash A > B} [R-]}{\Gamma \vdash A \multimap B} [R-\multimap]}{\Gamma \vdash \Delta} [Cut]$$

$[R\multimap]$ Symmetric.

$[R\mathfrak{A}]$ If the last contraction is a \mathfrak{A} contraction, the proof structure and reduction sequence look as follows.



Removing the par link and splitting the remaining conversions over the two substructures will give us the situation shown below.



We apply the induction hypothesis to obtain a sequent proof d_1 of $\Gamma \vdash A \circ B$ and a sequent proof d_2 of $A \wp B \vdash \Delta$ and combine these two proofs as follows.

$$\frac{\frac{\vdots d_1}{\Gamma \vdash A \circ B} \quad \frac{\vdots d_2}{A \wp B \vdash \Delta}}{\Gamma \vdash \Delta} [Cut]$$

The other cases are similar □

4 Complexity

In this section I will discuss the computational complexity of the contraction criterion for several different fragments of the proof net calculus.

4.1 Binary Without Structural Conversions

A first case is to decide the contractibility of a proof structure containing only binary links and without any structural conversions. When we look at the redexes of the different contractions, we see that there is no possibility of overlap: a link with three ports cannot be linked at two of its ports by two different links while having each of its nodes be at most once a conclusion and at most once a premiss of its link, as required by our definition of proof nets.

So even a naive contraction strategy which traverses the graph in search of contractible par links and contracts them as soon as they are found then makes another pass until it either fails to contract any par links — in which case the proof structure is not a proof net — or until there are no par links left — in which case we *do* have a proof net. This gives us an $O(n^2)$ algorithm, where n is the number of links in the graph. Without too much effort, we can improve this to $O(p^2)$ where p is the number of par links in the graph.

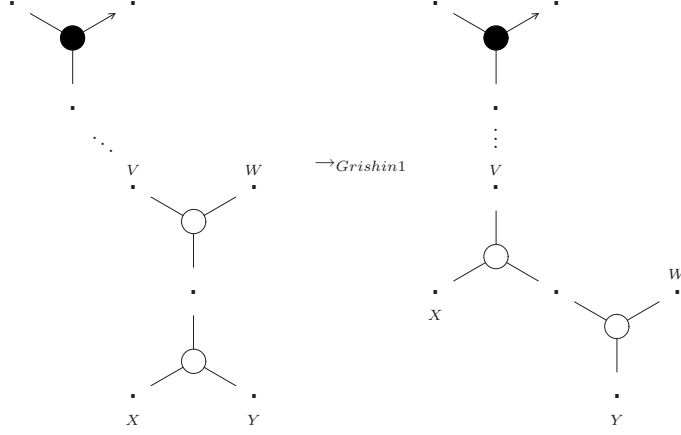


Figure 20: Moving a cotensor link left towards the corresponding par link

4.2 Binary With Grishin Interactions

A more complex case uses only the binary connectives but adds the Grishin rules of Figures 14 and 15.

Look at the $[L \multimap]$ contraction of Figure 8 and suppose the top and bottom part of the cotensor and par link are not already connected. For a Grishin interaction to apply, we need the dual residuated cotensor link to be connected to a residuated tensor link. There are two cases to consider. If we can reach the par link by the left branch of the tensor link, we apply the *Grishin 1* rule as shown in Figure 20. The cotensor links moves upward and to the left, reducing the distance to the par link.

Symmetrically, if the par link can be reached by the right branch of the tensor link, we apply the *Grishin 2* rule as shown in Figure 21. We move the cotensor link up and to the right and one step closer to the par link it needs to reach for its contraction.

When we spell out all different possibilities for all different par links, we end up with the schematic contractions shown in Figures 22 and 23 for the binary residuated and dual residuated connectives. $A|B$ indicating that either Grishin rule A or Grishin rule B applies, depending on the structure the tensor link finds itself in, and with $A.B$ indicating that we apply Grishin rule A followed by Grishin rule B . We remark that this sequencing operation means moving links first up then towards the C formula in the structure.

A second important point to note is that these operations can be *nondeterministic*. For example if both subnets of a generalized contraction contain links, we can use either possibility to move toward a contraction redex.

However, the situation changes when we separate the Grishin I and Grishin IV interactions. In the Grishin I situation, only the substructures containing just primed rules will remain. While this removes the non-determinism for the

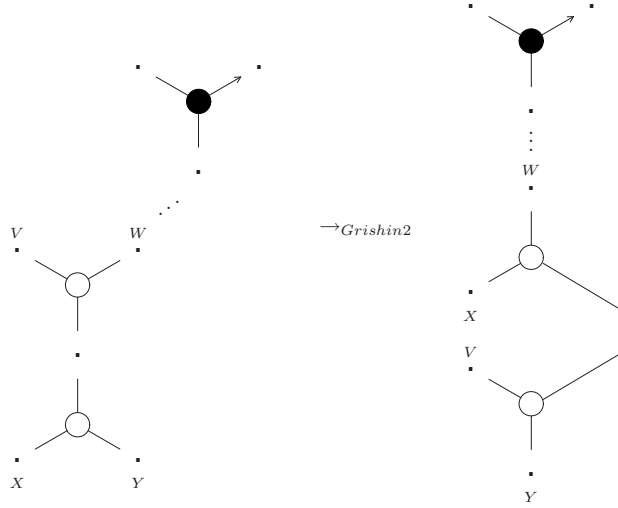


Figure 21: Moving a contensor link right towards the corresponding par link

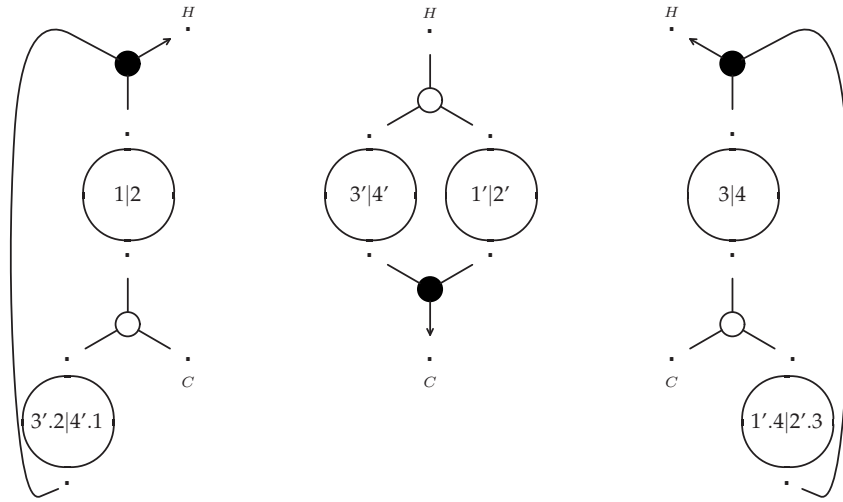


Figure 22: Contractions: binary dual residuated with Grishin

(co-)implications — only the ‘standard’ contractions are valid in this case — the product formulas will still potentially generate multiple solutions. In the Grishin IV situation, however, *all* non-determinism disappears.

The generalized contractions suggest the following algorithm for determining contractability in the Lambek-Grishin calculus: we use two disjoint set data

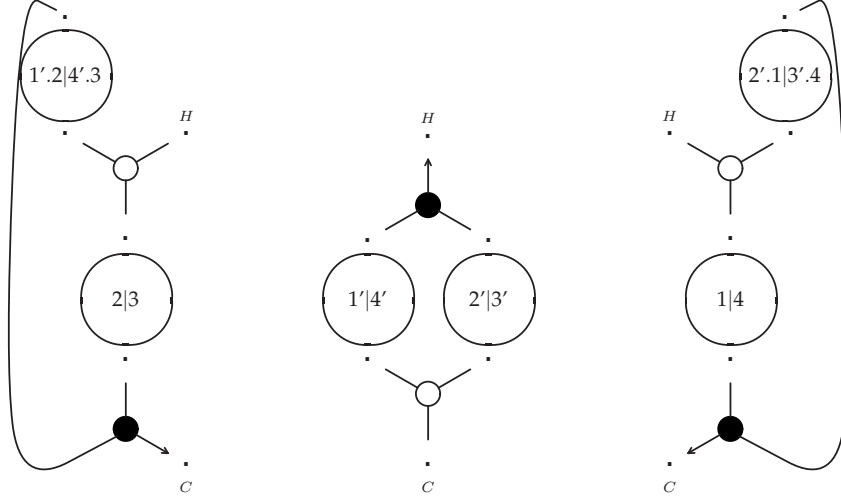


Figure 23: Contractions: binary residuated with Grishin

structures, one for residuated connected components of tensor links and one for dual residuated connected components of tensor links.

Now to determine contractability of the binary dual residuated connectives, it suffices to know that both hypotheses and conclusions of the two substructures in the figure are connected by a path of residuated tensor links. If they are, we perform a set union operation on the hypothesis and conclusion vertices in both disjoint set data structures. For the residuated connectives, we simply verify connectedness by a path of dual residuated tensor links.

In the absence of either the Grishin class I or the Grishin class IV structural rules, some of the substructures of the figure will be required to be empty, but this will not influence the complexity.

The total cost of deciding whether an abstract proof structure with v vertices, t tensor links and p par links is contractible is therefore summarized as follows.

Initialisation: v MAKE-SET operations and t UNIONS.

Contraction: at most $2p$ FIND-SET operations followed by two UNIONS.

We may still have to check all p par links to find one which is contractible, giving a total of p^2 'contraction attempts'. This gives us an upper bound up the total complexity of $\Theta(p^2 \log v)$ (Cormen, Leiserson & Rivest 1990, p. 449)², only a small increase over the naive solution without structural rules.

²This is the complexity when using just the path compression heuristic, the actual complexity is $O(m\alpha(m, n))$ where α is the 'inverse' Ackerman function

4.3 Binary and Unary Without Structural Conversions

For a binary contraction a tensor and a par link have to be connected at the two ports of the par links without the arrow. Given that there are only three ports to every binary link, this means it is impossible for two par links to both be candidates for reduction with the same tensor link.

With the unary contractions, this situation changes. Since only one port of the tensor and the par link have to be connected, a tensor link can be a candidate for reduction with two par links.

5 Applications

I will now turn to some applications of the Lambek-Grishin calculus LG. First by showing that then languages generated by LG grammars are outside the context free languages.

5.1 LG and Tree Adjoining Grammars

Kandulski (1988) shows that the non-associative Lambek calculus NL generates only context free languages. Several additions to NL have been proposed to increase the expressive power of the calculus. The solution advocated by Moortgat (1997) is a combination of modes, structural rules and control operators, whereas Baldridge & Kruijff (2003) propose modes and combinators.

More recent research (Moortgat 2007, Bernardi & Moortgat 2007) has looked at syntactic and semantic applications of the Lambek-Grishin calculus LG, which extends NL by adding dual residuated operators and interactions between the residuated and dual residuated operators, as shown in Figure 14 and 15.

It is the goal of this section to show that even LG with just the Grishin IV interactions can generate languages which are not context free. We will do this by giving an embedding translation of lexicalized tree adjoining grammars (LTAGs).

LTAGs are a widely used grammar formalism in computational linguistics (Joshi & Schabes 1996). The basic objects are trees and there are two operations on trees: *substitution*, as shown in Figure 24 replaces a leaf A^\downarrow by a tree with root A , whereas *adjunction*, as shown in Figure 25, replaces a node of the original tree by a tree.

An embedding of LTAGs into multimodal categorial grammars has been given in (Moot 2002, Chapter 10). I will improve on this result here. Most of the improvements are a consequence of using a subset of LTAGs which I will call LTAG_0 . This allows me to have just a single translation for adjunction points since all adjunction will take place at formulas with a negative polarity. In addition the number of structural rules is greatly reduced: instead of 12 different structural rules, the rules Grishin 1 and Grishin 2 will suffice.

I will first define the LTAG_0 grammars, discuss some of the differences with other definitions of LTAGs and then show that all interesting language classes

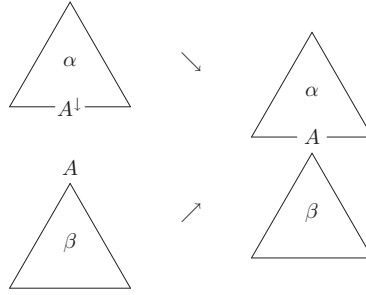


Figure 24: The Substitution Operation

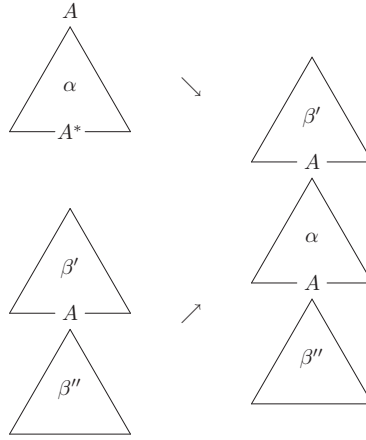


Figure 25: The Adjunction Operation

of LTAGs can be treated in LTAG_0 as well.

Definition 13 An LTAG_0 grammar is a tuple $\langle T, N_S, N_A, I, A \rangle$ such that

- T , N_S and N_A and three disjoint alphabets of terminals, substitution nonterminals and adjunction nonterminals respectively, we will use upper case letters A, B, \dots and of course the distinguished start symbol S to stand for members of N_S whereas we will use upper case letters T, U, \dots for members of N_A .
- I is a finite set of initial trees,
- A is a finite set of auxiliary trees.

The trees in $I \cup A$ are called the elementary trees.

Trees are subject to the following conditions:

- the root nodes of all initial trees are members of N_S ,

- the root nodes of all auxiliary trees are members of N_A ,
- every auxiliary tree has exactly one leaf which is a member of N_A which we will call the foot node,
- every elementary tree has exactly one leaf which is a member of T .
- every adjunction node, which we will mark as (T) in the tree, is on the path from the lexical leaf to the root of the tree.

This definition differs on several points from the standard definition of LTAGs as in for example (Joshi 1994). I will comment on each of these points.

Firstly, the difference between substitution and adjunction nonterminals is minor and is already implicit in the notation of A^\downarrow for substitution nodes and A^* for foot nodes, which like our choice of two different alphabets serves to remove any possible confusion about whether a substitution or adjunction operation should be applied to a node.

Some authors choose to mark null adjunction nodes explicitly. Given that the translation of the adjunction nodes is slightly more intricate than that of null-adjunction nodes, which we can just ignore, I have chosen opposition strategy of marking the (non-null) adjunction nodes explicitly.

Lexicalization is a fairly common restriction on LTAGs, our only additional restriction to it is that we require a *unique* terminal leaf.

The final and most important restriction is the requirement that every adjunction takes place on the path from the lexical leaf to the root of the tree. This is a real restriction, but one that simplifies our embedding result.

The definition of $LTAG_0$ is close to the definition of *normal* LTAGs used by Joshi, Shanker & Weir (1991) to show correspondence between LTAGs and combinatory categorial grammars, with the following differences: first, we don't need the requirement that all internal nodes have either the obligatory adjunction or the null adjunction constraint, second, the adjunction nodes are required to be on the path from the root to the lexical leaf instead of the foot node.

Definition 14 *Given an $LTAG_0$ grammar g a derivation tree d is a binary branching tree such that:*

- every leaf of d is an elementary tree of g ,
- every branch combines its two daughter trees using either the adjunction or the substitution operation,
- the root node is a tree which has the distinguished symbol S as its root and only terminals as its leaves.

We will show that there are $LTAG_0$ grammars that can handle all the interesting phenomena that LTAG grammars can.

Lemma 15 *The are $LTAG_0$ grammars generating*

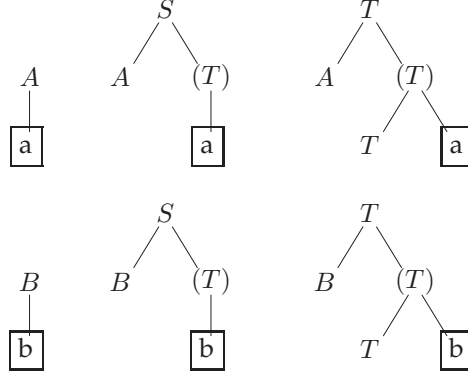


Figure 26: the copy language $\{ww \mid w \in \{a, b\}^+\}$

- the copy language $\{ww \mid w \in \{a, b\}^+\}$,
- counting dependencies $\{a^n b^n c^n\}, n > 0$ and
- crossed dependencies $\{a^n b^m c^n d^m\}, n > 0$

Proof Figure 26, Figure 27 and Figure 28 shows the LTAG_0 grammars generating the copy language, counting dependencies and crossed dependencies respectively. Showing we generate all and only the strings in the different languages is easy once we keep the following invariants in mind.

- For the copy language, the (T) adjunction point always has the entire copy of w as its descendants. Every adjunction adds either an a or a b to the end of the first string as well as to the end of the copy while creating a new adjunction point covering the new copy.
- For the counting dependencies, the (T) adjunction point always has the $a^n b^n$ part of the string as its descendants. Every adjunction will add an a and a b to both sides of the $a^n b^n$ part, while creating a new adjunction point containing the bigger $a^n b^n$ sequence. In addition, a c is added after the final b .
- For the crossed dependencies, first (T) will have all cs as its descendants, then (U) will have all cs and ds as its descendants. All (T) adjunctions generate both an a and a c keeping all cs under the (T) . When we generate the final a and c terminals, we start adjoining bs and ds at the (U) adjunction point, the bs appearing before the new adjunction point and the ds at the end inside it. \square

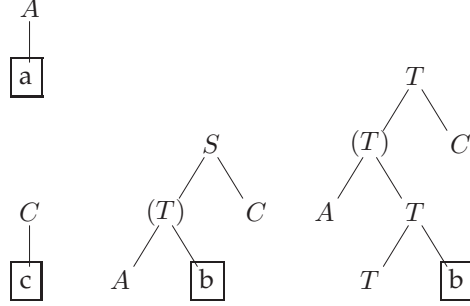


Figure 27: counting dependencies $\{a^n b^n c^n\}, n > 0$

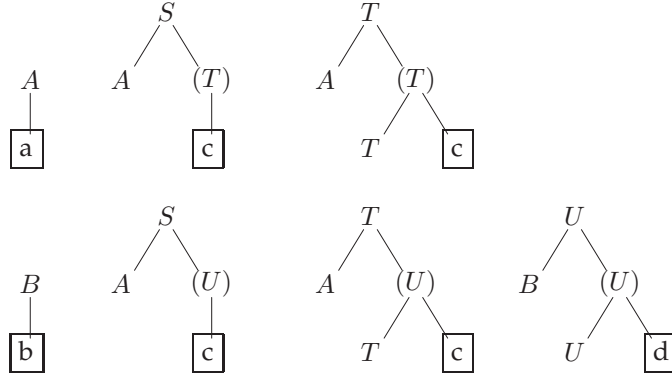


Figure 28: crossed dependencies $\{a^n b^m c^n d^m\}, n > 0$

Definition 16 Let g be an LTAG_0 grammar, we will define the corresponding LG grammar g' as follows. The atomic formulas A of g' will be $N_A \cup N_S$ to which we add a new formula i . We will translate all members of $N_A \cup N_S$ by the corresponding member of A , except for the foot nodes T , which will be translated as $T' := (T \circ i) \otimes i$.

We proceed by recursive descent of the LTAG_0 tree from the root R downwards towards the unique terminal leaf t .

- We start at the root R and use R as the current formula f .
- We translate an adjunction point (T) by assigning $f := (F \wp f) \wp ((T \circ i) \otimes i)$.
- We translate a binary branch by assigning $f := A \multimap f$ if the terminal leaf is a descendant of the right node or $f := f \multimap A$ if it is a descendant of the left node. A is a pure product formula representing the structure of the descendants of the other node; typically A is just an atomic formula.

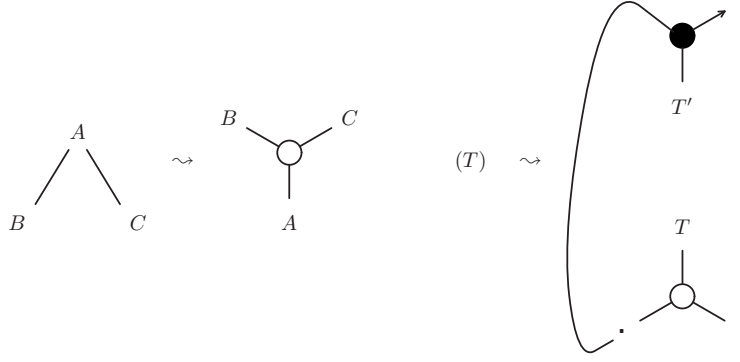


Figure 29: Translating LTAG_0 to LG

- When we arrive at the terminal leaf t , we add $\text{lex}(t) = R$ to the lexicon.

This translation is perhaps easiest to visualize in the form of trees as shown in Figure 29. From this figure it should be clear that the adjunction operation is going to correspond to two T axiom connections followed by the Grishin 1 and 2 rules and a contraction, as shown schematically in Figure 22.

Some properties to remark about this translation: first of all, internal nodes which are not adjunction points, that is have the null adjunction constraint, disappear. Given that they are just ornamental in the LTAG grammar, this is just a cosmetic change.

Secondly, given that the following sequent is derivable in LG even without interaction principles

$$(((T \circ -i) \otimes i) \leftarrow \wp A) \wp T \vdash A$$

we can handle the possibility that no adjunction takes place at an adjunction node very naturally.

Another point, made clear by Figure 29 is that our LTAG_0 trees are upside-down! This is easy to remedy using the symmetries of LG, but I feel the current solution using the residuated instead of the dual residuated connectives as the ‘main’ connectives is to be preferred.

A final point concerns the use of $(A \circ -i) \otimes i$ formulas. This is necessary to prevent multiple auxiliary trees to be adjoined at the same time, connecting the root and foot nodes together to form a sort of derived auxiliary tree. In all of the given example grammars this would lead to overgeneration: we explicitly want to assign the root and foot nodes a null adjunction constraint. In the LG grammar we exploit the derivability relation for this. Because $A \not\vdash (A \circ -i) \otimes i$ we

cannot perform the $[R\circ-]$ contraction when we connect the root and foot node of two auxiliary trees. There are other possible solutions: one is to use different formulas T and U for the foot and root nodes as well as for the corresponding atomic formulas in the adjunction point. This would correspond to an obligatory adjunction constraint similar to the obligatory adjunction solution proposed by Joshi et al. (1991), but it would result in added lexical ambiguity to allow for cases where there is no adjunction at an adjunction point. Another solution would be to use the unary modes to implement the same restriction and replace $(A\circ-i) \otimes i$ by $\Diamond A$.

As an example, the following LG lexicons correspond to the three LTAG₀ grammars we presented before.

The copy grammar is shown below.

$lex(a) \quad A$
 $lex(a) \quad (T \leftarrow (A \circ S)) \rightsquigarrow ((T \circ i) \otimes i)$
 $lex(a) \quad ((T \circ i) \otimes i) \circ (T \leftarrow (A \circ T)) \rightsquigarrow ((T \circ i) \otimes i)$
 $lex(b) \quad B$
 $lex(b) \quad (T \leftarrow (B \circ S)) \rightsquigarrow ((T \circ i) \otimes i)$
 $lex(b) \quad ((T \circ i) \otimes i) \circ (T \leftarrow (B \circ T)) \rightsquigarrow ((T \circ i) \otimes i)$

In the following, we will choose to abbreviate the formulas $(T \circ i) \otimes i$ by T' to improve the readability. The abbreviated version of the grammar above looks as follows.

$lex(a) \quad A$
 $lex(a) \quad (T \leftarrow (A \circ S)) \rightsquigarrow T'$
 $lex(a) \quad T' \circ (T \leftarrow (A \circ T)) \rightsquigarrow T'$
 $lex(b) \quad B$
 $lex(b) \quad B(T \leftarrow (B \circ S)) \rightsquigarrow T'$
 $lex(b) \quad T' \circ (T \leftarrow (B \circ T)) \rightsquigarrow T'$

The grammar for counting dependencies looks as follows.

$lex(a) \quad A$
 $lex(b) \quad A \circ ((T \leftarrow (S \circ C)) \rightsquigarrow T')$
 $lex(b) \quad T' \circ (A \circ (T \leftarrow (T \circ C))) \rightsquigarrow T'$
 $lex(c) \quad C$

Finally, the grammar for crossed dependencies is shown below.

$lex(a) \quad A$
 $lex(b) \quad B$
 $lex(c) \quad (T \leftarrow (A \circ S)) \rightsquigarrow T'$
 $lex(c) \quad T' \circ (T \leftarrow (A \circ T)) \rightsquigarrow T'$
 $lex(c) \quad (U \leftarrow (A \circ S)) \rightsquigarrow U'$
 $lex(c) \quad T' \circ (U \leftarrow (A \circ T)) \rightsquigarrow U'$
 $lex(d) \quad U' \circ (U \leftarrow (B \circ U)) \rightsquigarrow U'$

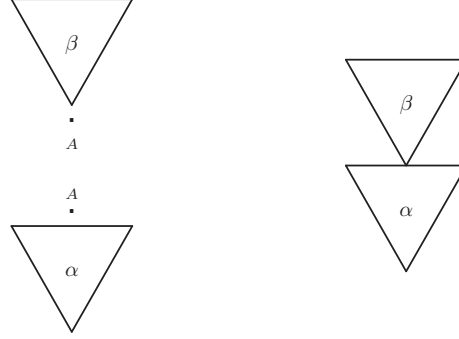


Figure 30: The substitution operation on abstract proof structures

Lemma 17 *Let g' be the translation of an LTAG_0 grammar g into LG and let S be the proof structure corresponding to an auxiliary tree of g . Any proof net \mathcal{P} of g' which has S as a substructure has the root and foot node of S connected to the two atomic formulas of one adjunction point of \mathcal{P} .*

Proof Look at the root node r and the foot node f of a proof structure S . Given that r and f are both members of N_A , there are few possibilities for identifying r with other atomic formulas. If we identify r with a foot node, we will end up with a non-contractible tree, given that $A \not\vdash (A \circ -i) \otimes i$. Connecting it to a substitution leaf is impossible, given that $N_S \cap N_A = \emptyset$. So the only possibility is to perform an axiom link with an adjunction node.

Now look at the corresponding foot node. If we identify it with the root of another auxiliary tree, we get a non-contractible tree again. The root of an initial tree is excluded, given that we have an atom from a different alphabet. Attaching it to a *different* adjunction node is excluded as well, since we won't be able to perform either $[L \multimap]$ contraction. So the only remaining option is to attach it to the other atomic formula of the same adjunction point as the root node. \square

Lemma 18 *For every LTAG_0 grammar g which generates language L , the LG grammar g' generates the same language.*

Proof (Sketch) Let d be an LTAG_0 derivation tree ending in tree t using a grammar g . Let g' the corresponding LG grammar. We show that g' derives the same tree t .

For every substitution we simply identify the two corresponding nodes in the LG proof structures. This will correspond to an axiom in the resulting proof net and produces a tree isomorphic to the result of applying the substitution

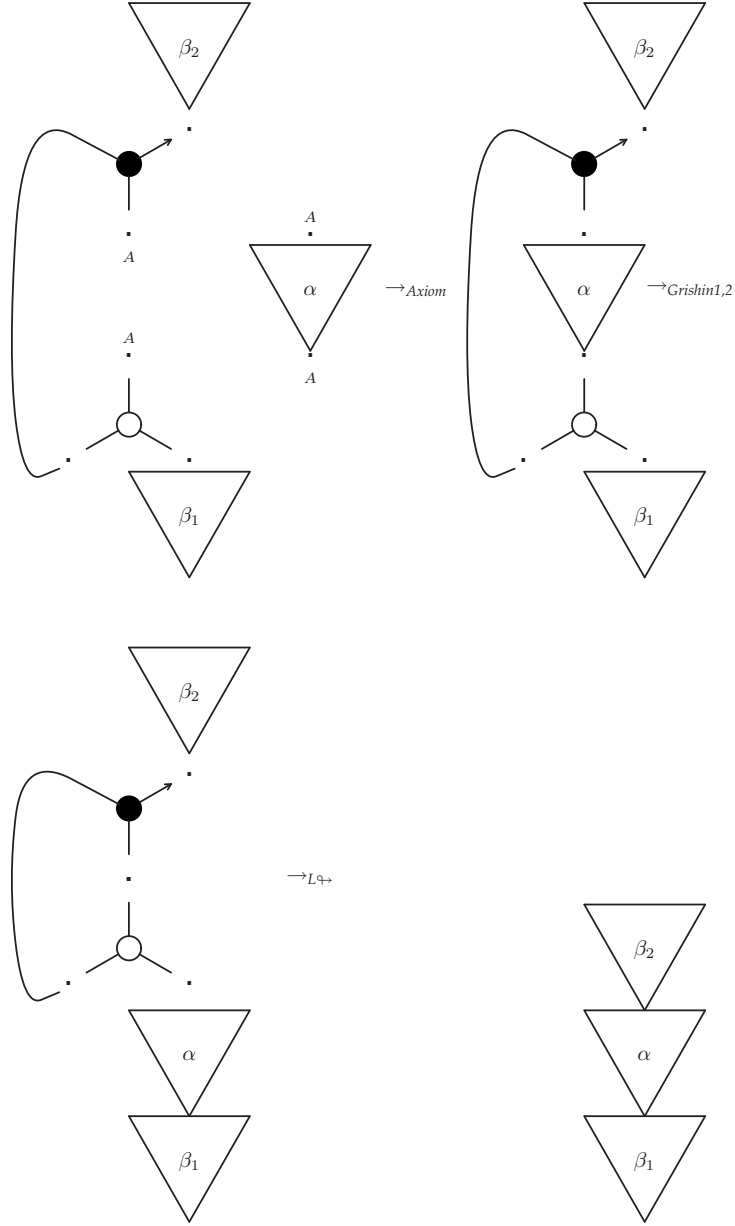


Figure 31: The adjunction operation on abstract proof structures

operation in g . Figure 30 shows how the (abstract) proof structures can be combined.

For every adjunction we identify the root and foot nodes with the two nodes of the adjunction point. We can apply the $[L\otimes]$ and the $[R-\circ]$ contractions

straight away. After all axiom connections are performed it is time for the generalized $[L \multimap]$ contractions, working from the inside out. Every generalized contraction will produce a subtree which is isomorphic to the result of applying the adjunction operation in g . Figure 31 shows the different operations on the abstract proof structure.

For the other direction, we show that if g' produces a derivation ending in tensor tree t , then g produces this same tree t as well.

Lemma 17 shows that axiom connections corresponding to adjunctions come in pairs and we need to apply rules Grishin 1 and 2 in order to contract the $[L \multimap]$ link in the case of adjunction at a node. Having no adjunctions at an adjunction point corresponds to connecting and contracting the T and T' substructures. All other axiom connections correspond to substitutions in the grammar \square

By giving an embedding translation of LTAGs, we've shown that LG with the Grishin class IV interactions handle more complicated phenomena than those we can treat using context free grammars. *How much* more is still unclear.

Moortgat's (2007) treatment of generalized quantifiers appears to move us beyond simple LTAG grammars. Generalized quantifiers are generally handles using multi-component tree adjoining grammars. Generalizing the above translation to MCTAGs seems an interesting possibility, whereas it would also be a candidate for giving an *upper bound* on the descriptive complexity of LG.

The type $(s \multimap s) \multimap np$ assigned to generalized quantifiers looks similar to the $(t \multimap a) \multimap t$ translation of insertion points. Both are instances of the $q(A, B, C)$ operator which would make NL+q another candidate for the descriptive complexity of LG.

5.2 Scrambling

6 Conclusions

I've shown how to extend the proof net calculus for $\mathbf{NL} \diamond_{\mathcal{R}}$ to display logic, adding several families of connectives while dropping only the constraint that a tensor link has a unique conclusion. I've shown basic soundness and completeness results and discussed the complexity of the contraction criterion for several sublogics.

Finally, I've shown how to embed LTAGs into LG, giving a lower bound on the descriptive complexity of LG and making the logic a candidate for a mildly context sensitive grammar.

Here	LL	DL	TLG	BLL
Structural — Nullary				
ϵ		Φ		1
Structural — Unary				
$\langle . \rangle$		\circ	$\langle . \rangle$	
$[.]$		\flat	\flat	
$\lceil . \rceil$		\sharp	\sharp	
Structural — Binary				
\circ	,	;	\circ	
$<$		$<$		
$>$		$>$		
\diamond				
\triangleleft				
\triangleright				

Table 6: Translation Key — Structural Connectives

A Translation Key

References

- Areces, C., Bernardi, R. & Moortgat, M. (2001), Galois connections in categorial type logic, in G.-J. Kruijff, L. Moss & R. T. Oehrle, eds, ‘Proceedings of FGMOL 2001’, Vol. 53 of *Electronic Notes in Theoretical Computer Science*, Elsevier, pp. 3–20.
- Baldrige, J. & Kruijff, G.-J. (2003), Multi-modal combinatory categorial grammar, in ‘Proceedings of the Tenth Conferences of the European Chapter of the Association for Computational Linguistics’, ACL, Budapest, Hungary, pp. 211–218.
- Bernardi, R. & Moortgat, M. (2007), Continuation semantics for symmetric categorial grammar, in ‘Proceedings of WoLLIC 2007’, Vol. 4567 of *LNCS*, Springer, pp. 53–71.
- Cormen, T. H., Leiserson, C. E. & Rivest, R. L. (1990), *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts.
- Girard, J.-Y. (1987), ‘Linear logic’, *Theoretical Computer Science* **50**, 1–102.
- Goré, R. (1998), ‘Substructural logics on display’, *Logic Journal of the IGPL* **6**(3), 451–504.
- Grishin, V. N. (1983), On a generalization of the Ajdukiewics-Lambek system, in A. I. Mikhailov, ed., ‘Studies in non-classical logics and formal systems’, Nauka, Moscow, pp. 315–334.

Here	LL	DL	TLG	BLL
Logical — Nullary				
1	1	1		1
\perp	\perp	0		0
Logical — Unary				
\diamond		\diamond	\diamond	
\square		\square	\square^\downarrow	
$\cdot 1$		$\cdot 1$	$\cdot 1$	$\cdot l$
$1 \cdot$		$1 \cdot$	$1 \cdot$	$\cdot r$
$\cdot \perp$	$\cdot \perp$	$\cdot 0$	$\cdot 0$	
$\perp \cdot$		$0 \cdot$	$0 \cdot$	
Logical — Binary				
\otimes	\otimes	\otimes	\bullet	\otimes
$\neg \circ$	$\neg \circ$	\rightarrow	\backslash	\backslash
$\circ \neg$		\leftarrow	$/$	$/$
\wp	\wp	\oplus		\oplus
\wp		\vee		
\wp		\wedge		
\swarrow				
\searrow				
\nearrow				
\nwarrow				

Table 7: Translation Key — Logical Connectives

- Joshi, A. (1994), Tree-adjoining grammars, *in* R. E. Asher, ed., ‘The Encyclopedia of Language and Linguistics’, Pergamon Press, Oxford, UK.
- Joshi, A. & Schabes, Y. (1996), Tree-adjoining grammars, *in* G. Rosenberg & A. Salomaa, eds, ‘Handbook of Formal Languages’, Vol. 3, Springer, New York, pp. 69–123.
- Joshi, A., Shanker, V. & Weir, D. (1991), The convergence of mildly context-sensitive grammar formalisms, *in* P. Sells, S. Shieber & T. Wasow, eds, ‘Foundational Issues in Natural Language Processing’, MIT Press, Cambridge, Massachusetts, pp. 31–82.
- Kandulski, M. (1988), ‘The equivalence of nonassociative Lambek categorial grammars and context free grammars’, *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* **34**, 41–52.
- Lambek, J. (1993), From categorial grammar to bilinear logic, *in* K. Došen & P. Schröder-Heister, eds, ‘Substructural Logics’, Oxford University Press, Oxford, pp. 207–237.

- Moortgat, M. (1997), Categorical type logics, *in* J. van Benthem & A. ter Meulen, eds, 'Handbook of Logic and Language', Elsevier/MIT Press, chapter 2, pp. 93–177.
- Moortgat, M. (2007), Symmetries in natural language syntax and semantics: the Lambek-Grishin calculus, *in* 'Proceedings of WoLLIC 2007', Vol. 4567 of *LNCS*, Springer, pp. 264–284.
- Moot, R. (2002), Proof Nets for Linguistic Analysis, PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University.
- Moot, R. & Puite, Q. (2002), 'Proof nets for the multimodal Lambek calculus', *Studia Logica* **71**(3), 415–442.